electrochem

MDPI

Article

Application of Machine Learning in Battery: State of Charge Estimation Using Feed Forward Neural Network for Sodium-Ion Battery

Devendrasinh Darbar and Indranil Bhattacharya *

Department of Electrical and Computer Engineering, Tennessee Technological University, Cookeville, TN 38505, USA; dudarbar42@tntech.edu
* Correspondence: ibhattacharya@tntech.edu

Abstract: Estimating the accurate State of Charge (SOC) of a battery is important to avoid the over/undercharging and protect the battery pack from low cycle life. Current methods of SOC estimation use complex equations in the Extended Kalman Filter (EKF) and the equivalent circuit model. In this paper, we used a Feed Forward Neural Network (FNN) to estimate the SOC value accurately where battery parameters such as current, voltage, and charge are mapped directly to the SOC value at the output. A FNN could self-learn the weights with each training data point and update the model parameters such as weights and bias using a combination of two gradient descents (Adam). This model comprises the Dropout technique, which can have many neural network architectures by dropping the neuron/mode at each epoch/training cycle using the same weights and biases. Our FNN model was trained with data comprising different current rates and tested for different cycling data, for example, 5th, 10th, 20th, and 50th cycles and at a different cutoff voltage (4.5 V). The battery used for estimating the SOC value was a Na-ion based battery, which is highly non-linear, and it was fabricated in a house using $Na_{0.67}Fe_{0.5}Mn_{0.5}O_2$ (NFM) as a cathode and Na metal as a reference electrode. The FNN successfully estimated the SOC value for the highly non-linear nature of the Na-ion battery at different current rates (0.05 C, 0.1 C, 0.5 C, 1 C, 2 C), for different cycling data, and at higher cut-off voltage of –4.5 V $Na^{+,}$ reaching the R^2 value of ~0.97–~0.99, ~0.99, and ~0.98, respectively.

Keywords: neural network; State of Charge (SOC) estimation; sodium-ion battery; dropout technique

Citation: Darbar, D.; Bhattacharya, I. Application of Machine Learning in Battery: State of Charge Estimation Using Feed Forward Neural Network for Sodium-Ion Battery. *Electrochem* **2022**, *3*, 42–57. https://doi.org/10.3390/electrochem3010003

Academic Editor: Masato Sone

Received: 25 December 2021
Accepted: 10 January 2022
Published: 11 January 2022

1. Introduction

A recent report from the International Energy Agency (IEA) on the Global Electric Vehicle (EV) Outlook 2020 [1] showed a surge in demand for electric mobility in the coming decade across the world. Stated Policies Scenarios, which incorporates existing government policies, has estimated the rise in global battery capacity from 170 gigawatt-hours (GWh) per annum in 2019 to 1500 GWh per annum in 2030, whereas the Sustainable Development Scenario projected the battery capacity demand to 3000 GWh/year in 2030, driven by rapid electrification and a rise in electric heavy-duty vehicles. In short, there is global pressure for implementing the policy to minimize CO_2 emissions, and increasing battery-powered electric vehicles will make a considerable contribution to achieve the target.

Batteries are one of the expensive and important components in the electric vehicle; therefore, they must be managed properly by electronics and software, i.e., having a reliable battery management system (BMS). A BMS maximizes the performance (power and energy) delivered by the battery and its service life and protects the battery pack [2,3]. To achieve this specific task in a BMS, a sophisticated algorithm is implemented on these specialized electronics. A BMS must be able to estimate two fundamental types of non-measurable battery-pack quantities: (1) states that change quickly (state of charge, diffusion voltage, hysteresis voltage) and (2) parameters that change slowly (cell capacities, resistances).

State-of-charge estimation is very important because it is an input to cell balancing and estimates the energy/power calculations.

Battery state of charge (SOC) is the ratio of residual capacity to the total capacity. SOC is like a dashboard fuel gauge that reports a value from "Full (100%) to Empty" (0%) [4]. Current gasoline engines have a sensor to gauge the gasoline level, but presently there is no such type of sensor to measure SOC in the electric vehicle. Accurate estimation of SOC produces benefits such as avoiding harming the cells from over/undercharging (longevity), producing excellent performance, enhancing overall power system reliability, and reducing the cost of the overall system.

Estimating SOC is difficult because it is time dependent, highly non-linear based on battery chemistry, and varies with temperature. Traditionally, there are two methods to estimate the SOC such as a voltage-based method (open circuit voltage (OCV)) [5] and current-based method (coulomb counting) [6]. These methods have their limitations, such as coulomb counting is easy to implement but integral charge results in accumulating SOC estimated error and the OCV method ignores the effects of impedance, diffusion, and hysteresis voltage. These techniques are replaced with some advanced methods such as Kalman filter (KF), Extended Kalman filter (EKF), equivalent circuit method, and neural network [4,7]. One of the most known algorithms is the EKF algorithm, which estimates the SOC value based on measured voltage, current, and previously estimated SOC. This algorithm is often tied with an equivalent circuit model or a lumped parameter model, which requires complicated parameter identification to represent the non-linear nature of the battery.

Neural network (NN) is a powerful tool in predicting any non-linear system. It does not require any accurate formula or equivalent circuit model/parameters to define the relationship between the battery parameters and the SOC. It estimates SOC from the historical data (current, voltage, temperature, impedance), which are fed to the network, which adapts to them accordingly. Traditionally, various neural network architectures contain a different number of neurons, hidden layers, and activation function to determine the dynamic properties of a battery, which are usually composed of an input layer, hidden layers, and an output layer [8–10]. Charkhgard et al. [11] used the combination of EKF and a neural network comprising 30 neurons in the hidden layer and Gaussian as the activation function to estimate the SOC with Root Mean Square error of <2% for Li-ion battery. Du et al. [12] estimated the SOC with a maximum error of <2% using 10 neurons in the hidden layer and a sigmoid as the activation function; however, as the test data were collected at a constant current rate, it made the model less robust. Tiwari et al. [13] showed a cascaded forward backpropagation network comprising two hidden layers with 18 neurons each and sigmoid as an activation function, which had an accuracy of 99.99% for estimating SOC in Na-ion battery. However, the model was less robust because test data used to determine the SOC were the next immediate cycle; for example, the model was trained for the first cycle and tested for a second cycle, which made it difficult to conclude what would be the accuracy of the SOC for the subsequent or later cycles. Chemali et al. [14] used an interesting deep neural network to determine the SOC for Li-ion battery for different temperatures but showed only one-cycle data with Mean Absolute Error (MAE) of 1.10% for data recorded at 25 °C and 2.17% for −20 °C.

The Current state of the art is the Li-ion battery (LIB) technology in electric vehicles because of high energy density, good capacity retention, and pre-occupied infrastructure, but procurement of metals required to build the LIBs is getting difficult. For example, the extraction process of lithium is difficult and expensive, and has price instability and resource concentration for cobalt metal makes it hard to rely only on one battery technology [15,16]. A sodium ion battery is considered as a better alternative to LIBs because of sodium's low cost, natural abundance, higher chemical diffusion coefficient, similar reaction mechanism, and better electronic conductivity. However, it has a lower reducing potential (−2.71 V vs. Standard Hydrogen Electrode (S.H.E) than Li (−3.04 V vs. S.H.E) [17].

This paper shows how a Feed Forward Neural Network (FNN) can accurately determine the state of charge (SOC) for a Na-ion battery comprising $Na_{0.67}Fe_{0.5}Mn_{0.5}O_2$ (NFM) as a cathode and Na metal as a reference electrode. The FNN shows the following novelties: (1) It can estimate SOC for the non-linear nature of a Na-ion battery that has a higher number of redox activities than a Li-ion battery; (2) A FNN uses voltage, current, and charge as inputs to the network and estimates the SOC without Kalman filter or equivalent circuit model; (3) A FNN can self-learn its weight using gradient descent and can be trained for a smaller number of epochs, which make the computation faster; (4) A FNN, once trained for different current rates (0.05 C, 0.1 C, 0.5 C, 1 C, and 2 C for four cycles each), at a specific voltage of 4.2 V vs. Na^+/Na, can estimate the SOC for a battery cycled at different cutoff voltages (4.5 V vs. Na^+/Na), for different current rates, and for various cycles (5th, 10th, 20th, and 50th cycles). This is unique because most of the trained models have tested data for the immediate cycle [14] or for only one cycle [13] to evaluate and understand the robustness nature of the model.

After a brief introduction, the second section will discuss how a feed forward neural network is designed for estimating the SOC. The third section presents how sodium layered cathode material was synthesized in house, fabricated in a coin cell, generating data that was collected for training/testing data from the battery station. In the fourth section, performance of the FNN is evaluated on a variety of test datasets.

2. Materials and Methods

2.1. Methods

The fundamental block of deep learning is an artificial neuron. The neuron takes an input $(x_1, x_2, x_3 \ldots . x_n)$ and, based on feature importance, each input is assigned some corresponding weight $(w_1, w_2, w_3 \ldots . w_n)$. This neuron takes an aggregate of weighted inputs, applies some function, and gives the output, as shown in Figure 1. For McCulloch Pitt's (MP) neuron model, input/output can only be Boolean and all weights are unity [18]. All the input are added together, since all the inputs are Boolean, which means counting the number of things that have a value of 1.

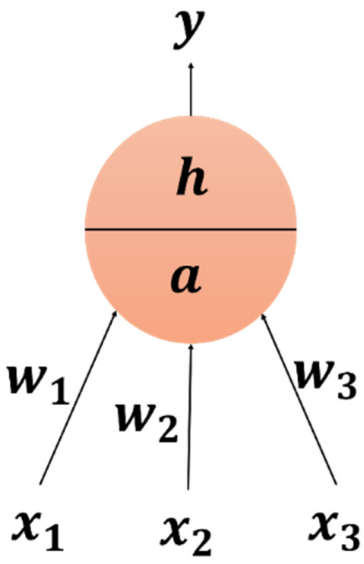

Figure 1. Artificial neuron model showing weights (w), inputs (x), pre-activation function as 'a', and activation function 'h' and 'y' as outputs.

Aggregation of this input can be called a pre-activation function 'a' (Equation (1)). The value of 'a' will pass through the function 'h', called the activation function, and gives output 1, which means the neuron will fire if the summation value is greater than some

threshold value 't', or it will output 0, which means the summation value is less than the threshold value (Equation (2)).

$$a(x_1\ ,\ x_2, x_3, \ldots . x_n) = a(x) = \sum_{i=1}^{n} x_i, \tag{1}$$

$$\begin{aligned} \hat{y} = h(a(x)) &= 1 \quad \text{if } a(x) \geq t \\ &= 0 \quad \text{if } a(x) < t \end{aligned} \text{ where } t \text{ is the threshold.} \tag{2}$$

If the predicted output '$\hat{y}$' is different from the true output 'y', then the error in this case would be the square of the difference between true and predicted values. The difference value is squared to avoid the cancelation of positive and negative difference values. Equation (3) shows the loss value for one bit of datum having numerous features, and Equation (4) shows the loss value of all the data points having different features of corresponding importance (weights).

$$\text{Loss/error } = (y_i - \hat{y}_i)^2 \tag{3}$$

$$\text{Loss/error } = \sum_i (y_i - \hat{y}_i)^2 \tag{4}$$

To minimize the error/loss value, the unique value of threshold 't' is used via the brute force method. With minimum loss value, the model is tested based on the accuracy it achieved (Equation (5)).

$$\text{Accuracy } = \frac{\text{number of correct prediction}}{\text{Total number of predictions}} \tag{5}$$

The MP neuron model divides the output into two sections: One section consists of the predicted value 1 and the other section consists of predicted value 0. The problem with this model is that it takes only binary values (0 and 1), has a poor learning algorithm to search the better threshold value 't', and it is a linear model.

To overcome the limitation of the MP neuron model, a Sigmoid neuron model can be used as an alternative with logistic function as an activation function 'h'. Pre-activation 'a' is the same as in the MP neuron model. Summation of the weighted inputs (could be n-dimensional) along with the bias as 'b' and output y as sigmoid or logistic function (Equation (6)). For the two inputs' case, the Sigmoid function is shown in Equation (7). For more than two inputs, the output equation is shown Equation (8). The input to the sigmoid neuron would be of any input value and output will be a continuous value between 0 and 1, for example, 0.4, 0.6, 0.8, and so on. The loss value (L) calculated for the Sigmoid neuron model would be the same as Equation (4).

$$\hat{y} = \frac{1}{1 + \ \mathrm{e}^{-(wx+b)}} \tag{6}$$

$$\hat{y} = \frac{1}{1 + \ \mathrm{e}^{-(w_1 x_1 + w_2 x_2 + b)}} \tag{7}$$

$$\hat{y} = \frac{1}{1 + \ \mathrm{e}^{-(\sum_i w_i x_i + b)}} \tag{8}$$

To minimize the loss value (L), the parameter of the sigmoid neuron model 'w_i' and 'b_i' should be such that the difference between predicted ($\hat{y}$) and true value (y) should be minimal. Initially, the model parameters are assigned some random values. Then, predict the output and compute the loss as in the Perceptron model [19]. It iterates by changing the value of 'w' and 'b' until the loss value is minimized. However, by changing the model parameters with some random guesswork, the loss value will decrease at some point and increase for the next potential value. In the actual world, we want to start at a random point and move towards the minimal loss value with some learning algorithm. Instead of guessing the model parameters, it requires a principal way of changing 'w' and 'b' in such

a way that loss value is minimized in the unique direction. With the help of the *Gradient Descent (GD)* rule [20], the values of 'w' and 'b' are updated by a partial derivative of loss function, which accounts for the entire data, computes the predicted output, computes the loss, and then updates the parameters again. This loop iterates continuously until good accuracy (loss value is minimal) is achieved. There are functions in the framework such as Pytorch [21] and Tensorflow [22], which compute the parameters automatically.

In the actual world, the data are not just linearly separable. Therefore, we need a complex function to fit the data. To have a complex function, using a simple sigmoid neuron model as the basic building block would not predict the output with high accuracy. Instead, combining several such sigmoid neurons in various layers, as shown in Figure 2 (known as Deep Neural Network (DNN)), can approximate a complex function between input and output [23,24]. The DNN would be differentiable, as the basic block is differentiable to learn the model parameters. The final output ($\hat{y}$) would be a function of $(x_1, x_2, x_3 \ldots . x_n)$ and it would be very complex because each input is passing through many neurons having an activation function with multiple transitions in different layers. With different network architecture, the one that gives the minimal loss value would be the best DNN approximating the relationship between inputs and the output. In summary, a neural network (Deep Neural Network (DNN)) with a certain number of hidden layers, an activation function, s number of neurons, and a learning rate could approximate any functions that exist between inputs and output.

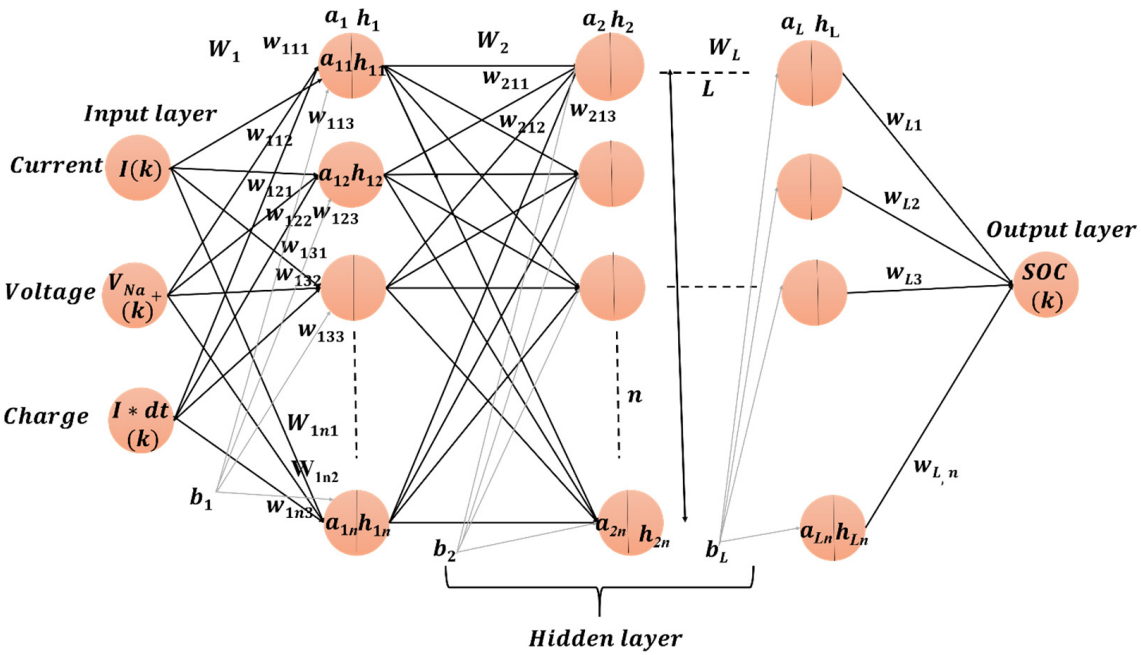

Figure 2. Architecture of a Deep Neural Network with L number of layers and n number of neurons at each layer. Weight and bias are associated to the corresponding layer and a detailed explanation is given in the main text. The input data are given as current, voltage, and charge at each time step and output of DNN estimates the *SOC* at every time step.

In Figure 2, the very first layer is known as the input layer, comprising current, voltage, and charge, and the last layer is known as the output layer for predicting the state of charge (SOC). All the other layers between input and output layers are known as intermediate/hidden layers. Each neuron has two things: One is the pre-activation, denoted as 'a', and the other is activation, denoted as 'h'. As in the case of the simple sigmoid neuron, the aggregation of inputs is known as pre-activation, and activation passes the aggregation of inputs to the sigmoid/logistic function. The weight is labeled as w_{ijk} where i = layer number, j = neuron number, and k = input number. For example, w_{121} is the first layer, the second neuron is attached to the first input, a_{ij} is the pre-activation, and h_{ij} is the activation

function of each neuron, where i is the layer number, j is the neuron numbers, and b_L is the bias associated with L number of layers. Below, Equations (9) and (10) show the matrix of weight 'W_1' and activation function 'h_1' for the first layer, respectively. The 'a_{1n}' is the pre-activation function for the first layer for n neurons (Equation (11)).

$$W_1 = \begin{bmatrix} w_{111} & \cdots & w_{11k} \\ \vdots & \ddots & \vdots \\ w_{1j1} & \cdots & w_{1jk} \end{bmatrix} \tag{9}$$

$$h_1 = \begin{matrix} h_{11} \\ h_{12} \\ \vdots \\ h_{1n} \end{matrix} \tag{10}$$

$$= w_{1j1} * x_1 + w_{1j2} * x_2 + w_{1j3} * x_3 + w_{1j4} * x_4 \ldots\ldots\ldots + w_{1jk} * x_k \ + b_{1,j} \tag{11}$$

For the second layer, output of the activation function of the first layer becomes the input for the pre-activation function of the second layer with the corresponding weights and bias. The general equations for 'L' number of layers are mentioned below, in Equations (12)–(14).

The pre-activation at layer 'i' is given by

$$a_i(x) = W_i h_{i-1}(x) + b_i \tag{12}$$

The activation at layer 'i' is given by

$$h_i(x) = g(a_i(x)) \tag{13}$$

where g is called the activation function.

The activation function at the output layer 'L' is given by

$$\hat{y} = f(x) = h_L = O(a_L), \tag{14}$$

where 'O' is called the output activation function.

The estimated output $\hat{y}$ will be a very composite and complex function of all the inputs passing through lots of non-linearities all the way. Once we compute the loss value, we can feed it to the function in the framework (from Pytorch, TensorFlow), which will update the parameter via backpropagation to minimize the overall loss value. The above network is also known as a Feed-forward network (FNN).

There are many architectural designs for a FNN based on different variables. They can be different depending upon the activation functions such as those chosen for sigmoid, tanh, ReLU, or LeakyReLU. The number of hidden layers can be changed to 2, 4, 5, and so on, the number of neurons in each layer can be 15, 16, 20 . . . etc., the learning rate can be 0.1, 0.01, 0.0001, etc., different batch sizes can be 16, 32, 128, etc., and different gradient descent techniques such as Adam, Adagrad, RMS prop, Momentum GD, etc. can be used. Distinct designs of a FNN determine the different loss values. This is known as hyper-parameter tuning and it can be performed using GridSearchCV [25] or RandomSearchCV [26]. In GridSearchCV, all the combinations of each variable with all other parameters are used to design the FNN and are run for a specific number of epochs. (Note: A full one-training epoch is considered when it includes one forward pass and one backward pass, the process of sending the Loss value signal backward to update the weights and bias.) Once all the combinations are run, they will give the best possible parameter whose loss value will be minimal. Similarly, in RandomsearchCV, it will randomly choose a variable combination for a predetermined number of combinations and provide the best combination parameter whose loss value is minimal. Hyper-parameter tuning via GridSearchCV is computationally time consuming and gives out the minimal loss value. However, in machine learning,

getting the minimal loss value of training data does not guarantee the best model, as the model needs to be validated on the validation data. If validation loss for validation data is higher for the same model, which has minimal loss value on training data, then the model is encountered with the over-fitting. It occurs when the gap between the training loss value and validation loss is higher.

In this work, we designed a two-layer neural network with 15 neurons in each layer and used sigmoid as the activation function. Before data were fed to the neural network, the data were normalized using MinMaxScaler (Equation (15)).

$$x_{scaled} = \frac{x - min(x)}{max(x) - min(x)} \tag{15}$$

A few optimization techniques were used in this model such as a batch size of 16 and Adam. Traditionally, the model looks at all the data points, computes the partial derivative of losses for all the data points, and updates the parameters; this is computationally expensive. Using data points as Batch 'B', means B data points are fed to the network, which computes the partial derivative, keeping a log for all such B number of data observed, and updates the weights and bias accordingly. After all the data points in batch size are fed to the network it is considered as one epoch. Instead of updating the model parameters once, its update the weights and bias 'B' number of times. The Adam algorithm is the combination of the two-Gradient Descent (GD) rule (Equation (18)), which is a momentum-based GD (a history component is used to make the current update (Equation (16)), and the RMS prop GD (in which history is used to update the learning rate (Equation (17)) [27].

$$m_t = \beta_1 * v_{t-1} + (1 - \beta_1)(\nabla w_t) \tag{16}$$

$$v_t = \beta_2 * v_{t-1} + (1 - \beta_2)(\nabla w_t)^2 \tag{17}$$

$$w_{t+1} = w_t - \frac{\eta}{\sqrt{v_t} + \epsilon} m_t \tag{18}$$

The β_1 and β_2 are the exponential rates for the first and second momentum estimates, respectively, and have values less than 1: For example, β_1 : 0.9 and β_2 : 0.999. The η is the learning rate and ϵ is a tiny number to prevent any denominator from going to zero, for example, 10^{-8}.

To avoid the symmetry breaking problem for using similar weights, an initialization technique is used for the initial weights. Based on the activation function, an initialization technique was used; for example, the best initialization technique for the sigmoid and tanh functions is Xavier uniform and for ReLU/Leaky ReLU, the He initialization technique is preferred. For the FNN, weights were initialized using the Xavier uniform distribution with sampling interval [−r,r] (Equation (19) shows the equation of r) [28].

$$\text{where } r = \sqrt{\frac{6}{n_{in} + n_{out}}} \tag{19}$$

The n_{in} and n_{out} are the number of input and output connections, respectively.

Dropout Technique

The process to mitigate over-fitting is called regularization, which means modifications are made in the learning algorithm with the goal of reducing the generalization error/validation loss rather than training error. A few techniques are early stopping [29], data augmentation, L2 regularization [30], batch normalization, and dropout technique [31]. In early stopping, the number of epochs is stopped when the validation loss is minimal, whereas in data augmentation, more training data or defects' data are added for training the model. Among all, the dropout technique is an interesting way to minimize the validation error.

Consider the example of using 10 different model architectures to approximate the relation between input and output. Instead of relying on the output of one model, we could rely on the output of all 10 models by averaging all the output data. Training the data on all different models or with a different subset of data having a different number of neurons, layers, or activation functions and then computing the loss value for all such neural networks will be computationally expensive. Instead, we can build a model in such a way that it shares the same weights and bias for neurons but has a different number of hidden layers and neurons and gets updated only when it is necessary. This configuration can be achieved via the Dropout technique where a neuron can be dropped based on some threshold value. For example, if the value of a node is 0, that means the node/neuron is dropped, and if it is 1, then keep the node/neuron in the network. If the neural network comprises 15 neurons, the combination of the network is 2^{15}. For n neurons, 2^n different possible NN architectures can be designed. For the dropped-out architecture, the weights for the nodes are kept in the network, which is going to be same as the original network. The model moves according to the previously fed data to the network, calculates the loss values, backpropagates the loss values, and updates the weights, which are used to compute the output value, but weights that are connected to the dropped node/neuron are not updated. For the next architecture, if the dropped neuron is connected, it will update the weights from the last iteration. In this way, weights and bias values are propagated or shared throughout multiple architectures, making them computationally workable and less time consuming. Each neuron will be present in half of all the networks and, thus, it will be updated for 50% of the period during training. We used the dropout rate of 20% between the inputs and hidden layer, meaning one in five inputs will be randomly excluded from each layer update.

2.2. Material Synthesis

P2-type $Na_{0.67}Fe_{0.5}Mn_{0.5}O_2$ (NFM) was synthesized using the sol-gel technique. CH_3COONa (10% excess), $Fe(NO_3)_3.9H_2O$, $(CH_3COO)_2Mn.4H_2O$, and citric acid as cheating agent were dissolved in deionized water with an appropriate molar ratio. The mixed solution was heated at 80 °C and stirred until the deionized water was evaporated. The dried powder was ground and heated at 400 °C for 4 h. in air (ramp rate 5 °C/min) followed by subsequent heating at 950 °C for 15 h. (in air at ramp rate 5 °C/min). The final calcined powder was stored in an Argon-filled glove box (H_2O, $O_2 \leq 0.1$ ppm) to avoid exposure to humidity and air

2.3. Material Characterization

X-Ray Diffraction (XRD) was performed using a Rigaku Ultima IV diffractometer with D/tex Ultra High Speed Detector and PANalytical powder diffractometers over the 2θ range from 10° to 80° with a scan speed of 2°/min with Cu Kα radiation (power setting 40 kV, 44 mA). Crystallographic evaluations of the sol-gel synthesized P2-type $Na_{0.67}Fe_{0.5}Mn_{0.5}O_2$ were performed using the XRD patterns, shown in Figure 3a. The patterns showed that the NFM powder samples had a hexagonal, layered structure with a P63/mmc space group, as reported in our previous paper [32,33]. The morphology of the powder samples was observed using an ultra-high-resolution Field Emission Scanning Electron Microscope (FE-SEM) Hitachi SU7000. The particle shapes of the powdered samples were hexagonal crystalline having average particle sizes between ~0.8–2.5 μm (Figure 3b).

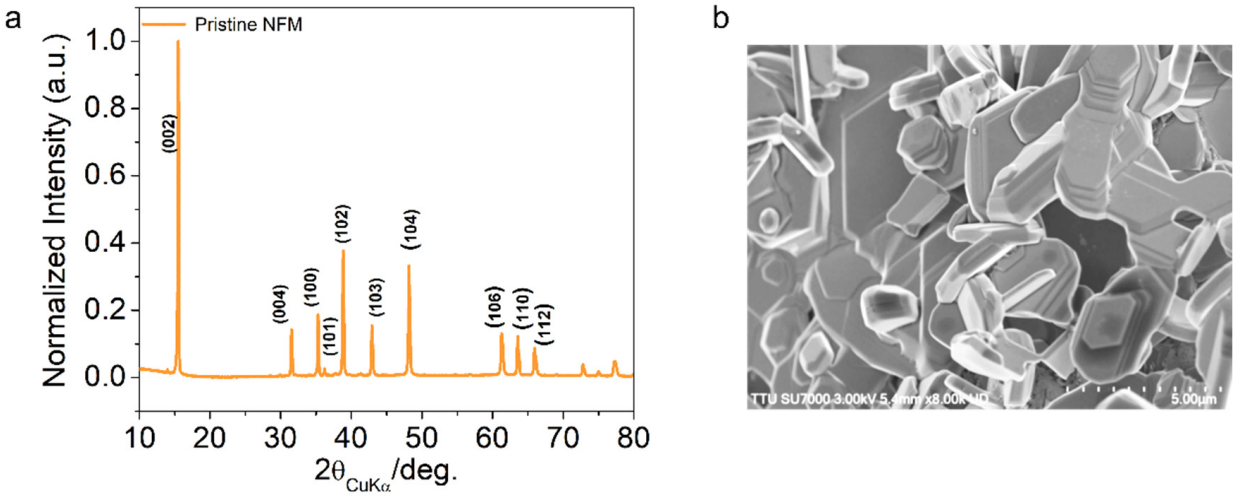

Figure 3. (**a**) X-ray Diffraction (XRD) patterns for $Na_{0.66}Fe_{0.5}Mn_{0.5}O_2$ (NFM) and corresponding hkl values. (**b**) The Scanning electron microscopy (SEM) image of NFM.

2.4. Electrochemical Characterization

Electrochemical performance was analyzed by fabricating the cathode (NFM) in a CR2032 coin cell ((0.787-inch diameter * 0.125-inch height), United Minerals and Chemical Corporation). The half-cells were assembled using Na metal (dia.: 7/16 inch) as the counter electrode, two glass microfibers (Whatman DBS 30, dia.: 5/8 inch) as the separator, and a slurry cast cathode (NFM) as the working electrode (dia.: 7/16 inch). The active material slurry was prepared by mixing the active material, Super P binder (Kynar PVDF), in the mass ratio of 80:10:10, respectively. N-Methyl-2-pyrrolidine (NMP) was used as a solvent for making the viscous slurry. Cathodes were prepared by casting the active material slurry on a carbon-coated aluminum current collector. The active material loading was 2–3 mg/cm^2), followed by drying under vacuum at 90 °C overnight. The electrolyte used was 1.0 M $NaClO_4$ in Propylene Carbonate (PC) with 2% Fluoroethylene Carbonate (FEC) as an additive. The coin cells were assembled in an Argon-filled glove box (H_2O, $O_2 \leq 0.1$ ppm). Galvanostatic cycling assessments were performed using a Maccor Series 4000 battery tester.

2.5. Data Used for Training and Test Data

2.5.1. Training Data

For training the FNN model, the data collected comprised current, voltage, and charge value at each time step, which were used as inputs to the FNN, and the calculated SOC was found at the output layer. The current, voltage, and charge values were found from the data set obtained from cycling the Na-based cathode coin cell (the Fabrication of the coin cell is explained in Section 2.4). Figure 4a shows the cycling behavior of the NFM cathodes cycled between 1.5–4.2 V vs. Na/Na^+ for five cycles each at different C-rates from 0.05 C to 2 C rate (1 C = 260 mAh/g). The first four discharge cycles for each C-rate were added together in the Excel file and used as the training data. The training data were divided into training and validation data for tuning the parameters. The validation data comprised 10% of the training data.

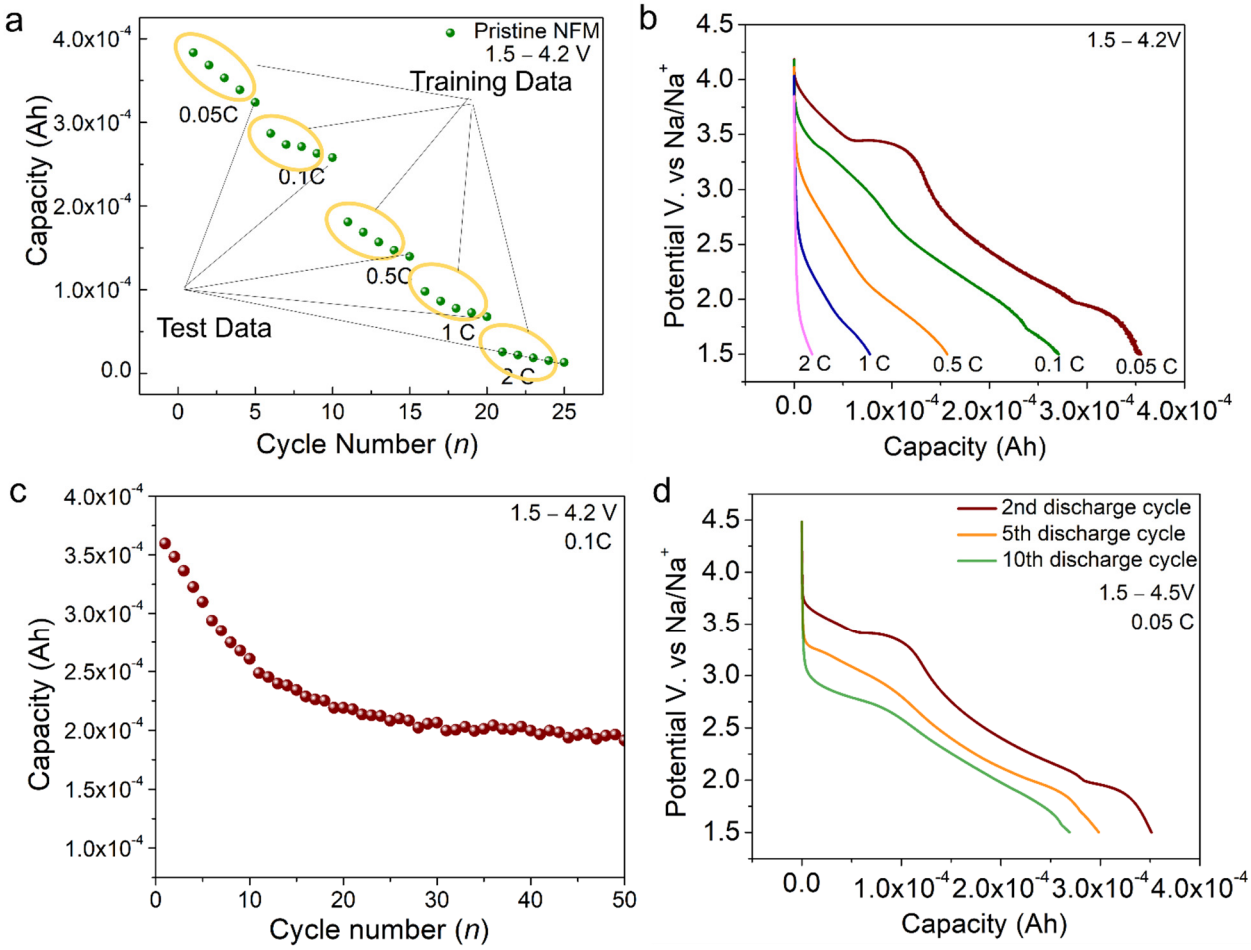

Figure 4. (**a**) Rate capability at different current rates from 0.05 C–4 C for NFM cathodes. (**b**) Galvanostatic discharge curve for NFM cycled between 1.5–4.2 V vs. Na/Na^+ for the fifth cycle of each current rate from 0.05 C to 2 C rate. (**c**) Specific capacity as a function of cycle number for NFM cathode cycled between 1.5–4.2 V vs. Na/Na^+ at 0.1 C rate for 50 cycles. (**d**) Galvanostatic discharge curve for NFM cycled between 1.5–4.5 V vs. Na/Na^+ for the 2nd, 5th, and 10th cycles at 0.05 C rate (1 C = 260 mA/g).

2.5.2. Testing Data

The testing data were characterized into three sections. (1) Figure 4b shows the charge–discharge profile curve of the NFM cathode cycled between 1.5–4.2 V vs. Na/Na^+ for different C-rate. (Note: The fifth cycle of each C-rate such as the 5th, 10th, 15th, 20th, and 25th cycles in Figure 4a). (2) Figure 4c shows the cycling data of the NFM cathode cycled between 1.5–4.2 V vs. Na/Na+ for 50 cycles at 0.1 C rate, out of which the 5th, 10th, 20th, and 50th cycles were used for testing the FNN model. (3) Figure 4d shows the charge and discharge voltage profile curves for the NFM cathode cycled between 1.5–4.5 V vs. Na/Na+ at a 0.05 C rate.

3. Results and Discussion

After training the FNN with the training data, as mentioned in Section 2.5.1, the FNN was tested for various test data sets fetched at different current rates, cycling data, and cutoff voltages, which were not part of the training data. The model was trained for 20 epochs, for a batch size of 32, a learning rate of 0.001, using a sigmoid activation function, using initialization technique as the Xavier uniform, and applying a dropout function to only the first hidden layer.

The computational speed for this training took a few minutes to train the model. Figure 5 shows the training and loss values as a function of epochs. Figure 6 shows the relationship between the different current rates for voltage and the state of charge. The true value of the SOC (solid line) was compared with the predicted SOC (dashed line) value for the tested data. The Accuracy between the true and estimated SOC values was measured using the R^2 value. Most of the report showed the graph for a time vs. SOC graph, as shown in Figure 6a, but it failed to give a more detailed information compared to Figure 6b. Thus, in this paper, most of the data was analyzed for voltage vs. SOC values. The R^2 value for the test data at 0.05 C (Figure 6b) showed 0.9960, which means 99.60% of the predicted value matched with the true SOC value. (Note: Though the R^2 value was above 99%, some positive values compensated for negative values, which failed to interpret the true accuracy. The true understanding of the SOC estimation is known when the Voltage vs. SOC graph was plotted for True and Estimated SOC values.) The Model mostly predicted the correct value of the SOC at the slope, with a slight variation at the plateau region between 40% and 85% of the SOC. At higher C-rates, the R^2 value decreased, for example, at 0.5 C (0.9874), 1 C (0.9747), and 2 C (0.9780), shown in Figure 6d–f. Figure 7 shows the graph of Voltage vs. SOC for the test data run at a 0.1 C rate for a different cycle (5th, 10th, 20th, and 50th), and the model performed much better in estimating the SOC value, having the R^2 value of ~0.99–0.97. The battery degraded when it was cycled for a longer period because of crystal structure instability, an increase in the impedance by forming a passivation layer on the electrode, the decomposition of the electrolyte, and many more reasons, as explained in our previous paper [32]. However, the model overcame the degradation nature of the battery and estimated the SOC value for a higher cycle (50th cycle) with good accuracy, of ~0.99, making the model more robust. Figure 8 shows the result of the estimated SOC value compared to the true value for the NFM cathode cycle at a higher cutoff voltage (4.5 V). The Model was never trained for the higher voltage, but it showed better accuracy, of ~0.994 for the 2nd, 0.9916 for the 5th cycle, and 0.9840 for 10th cycle. The battery run at higher cutoff voltage showed a different performance in terms of capacity and stability of the crystal structure. The SOC value differed when it was cycled at a higher potential, but the model predicted the SOC value with good accuracy, of greater than 98%. In summary, the model estimated the SOC value with an accuracy of ~0.98–~0.99 for a higher cutoff voltage (4.5 V), ~0.99 for a higher cycle number, and 0.97–0.99 at different current rates of the test data.

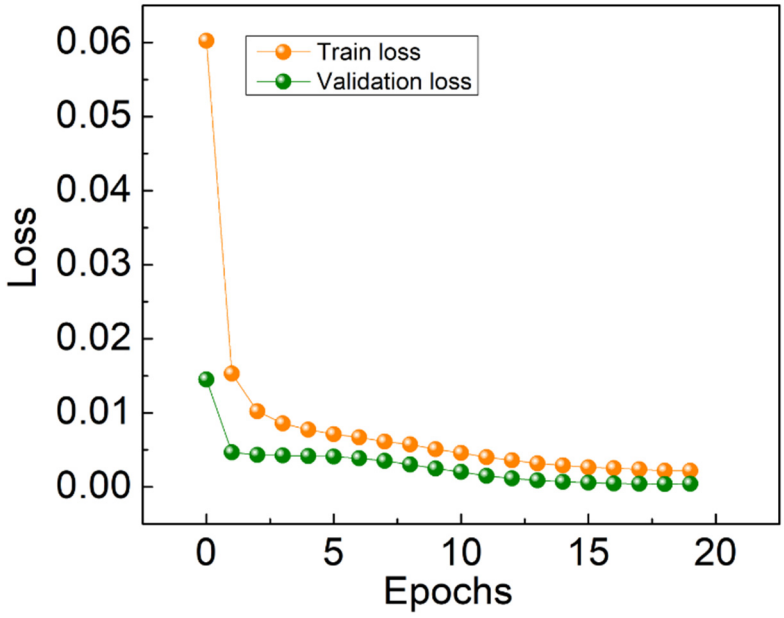

Figure 5. Loss value vs. number of epochs run using the Feed forward neural network.

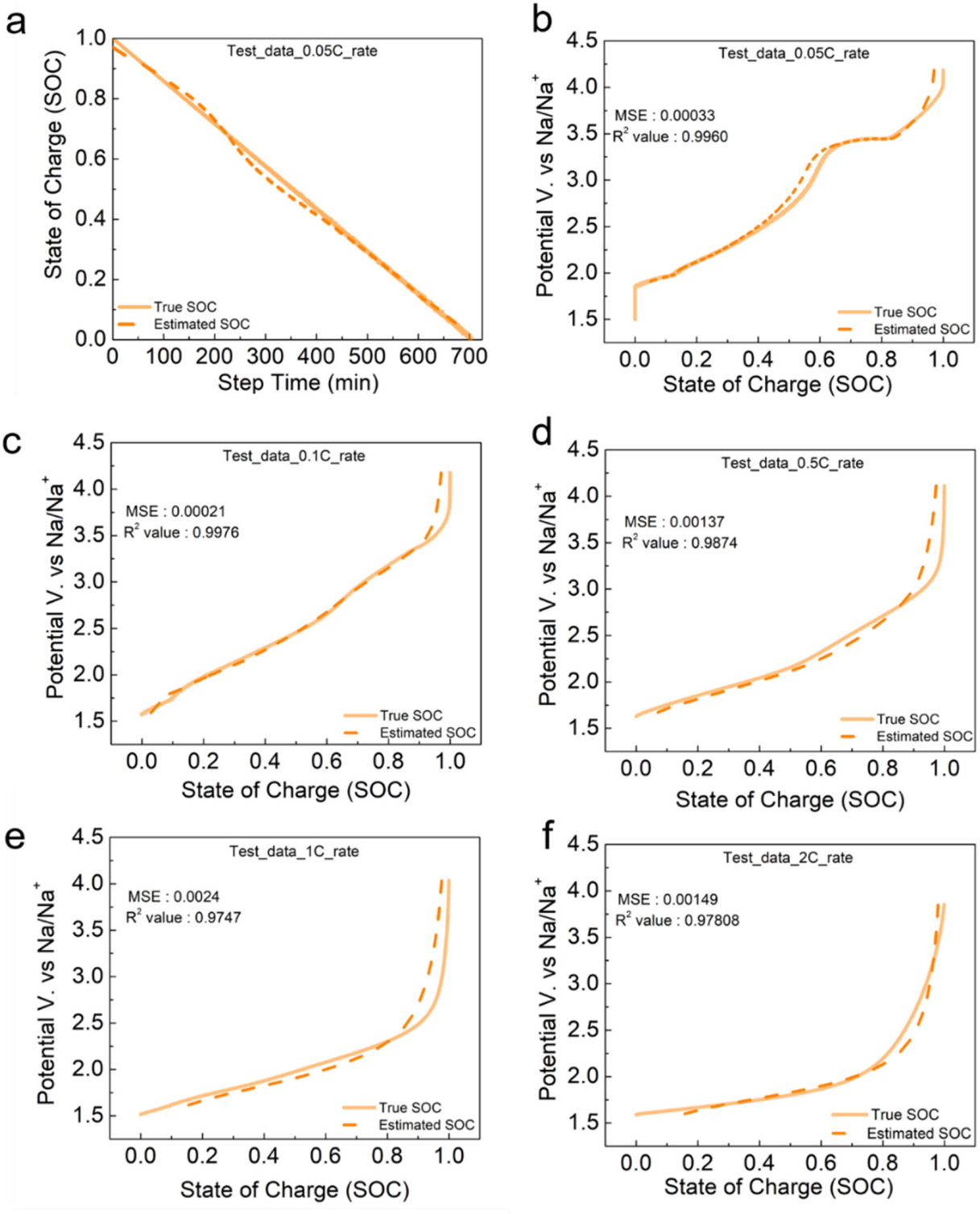

Figure 6. (**a**) Model performance is shown, using State of charge vs. time for the battery discharge data at a 0.05 C rate, by comparing the true and estimated values. However, for getting detailed information, the performance of the Feed forward neural network (FNN) model is shown, using voltage vs. State of charge graphs. Data used as battery discharged cycled between 1.5–4.2 V vs. Na/Na^+ at (**b**) 0.05 C, (**c**) 0.1 C, (**d**) 0.5 C, (**e**) 1 C, and (**f**) 2 C. The Mean square value and R^2 value are mentioned on each graph.

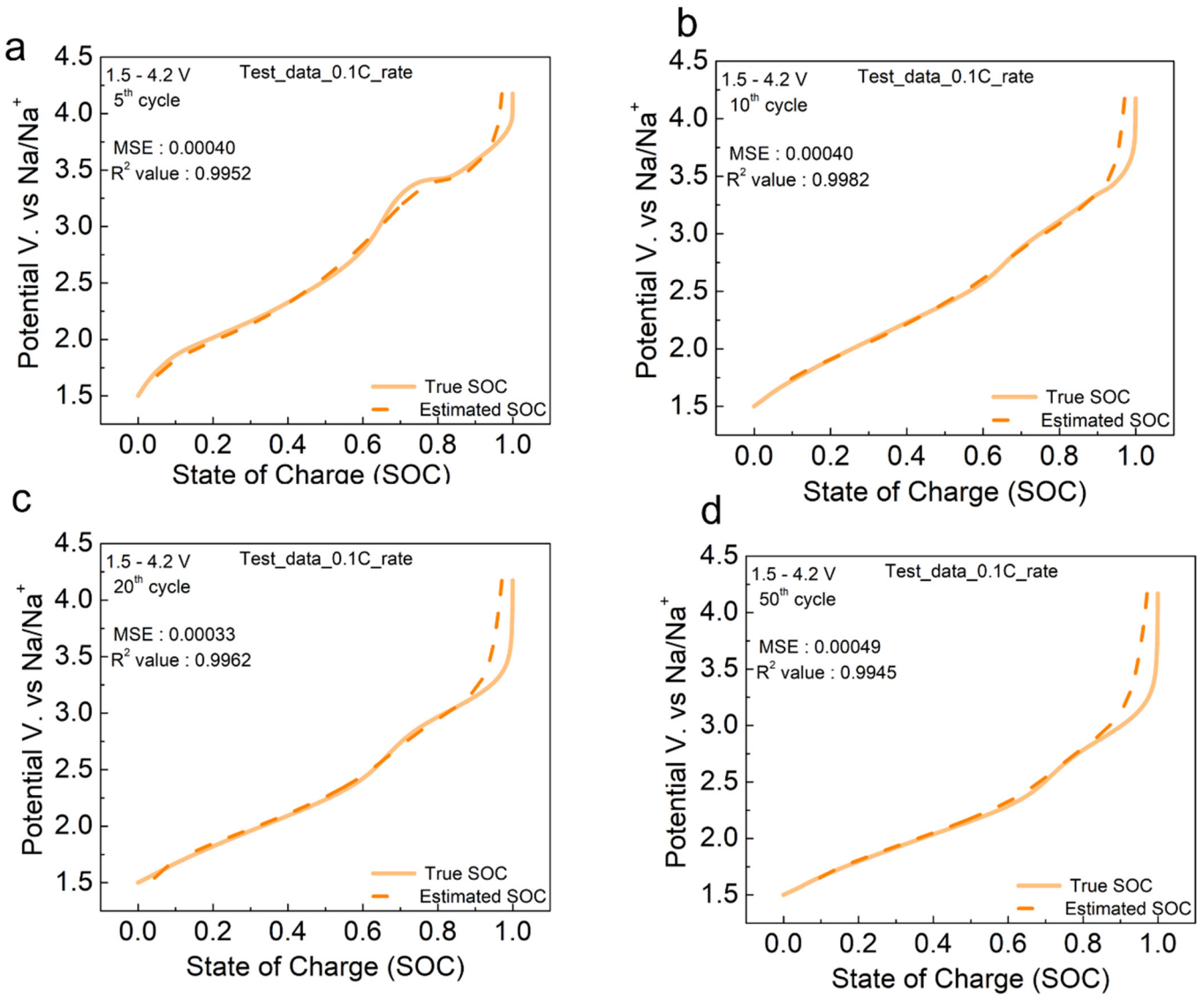

Figure 7. Performance of the FNN model, shown using voltage vs. State of charge graph rate by comparing the true and estimated values. Data used were from the battery discharged between 1.5–4.2 V vs. Na/Na^+ at a 0.1 C rate for the (**a**) 5th cycle, (**b**) 10th cycle, (**c**) 20th cycle, and (**d**) 50th cycle. Mean square value and R^2 value are mentioned on each graph.

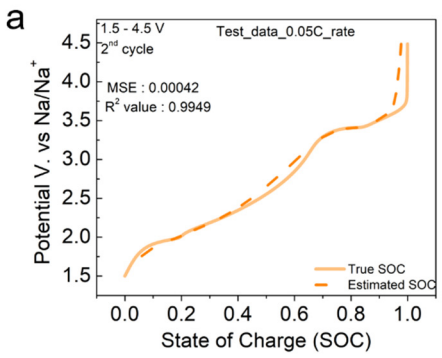

Figure 8. *Cont.*

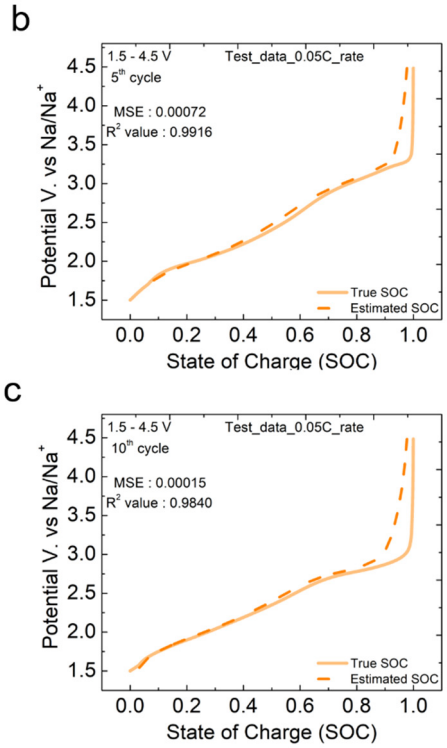

Figure 8. Performance of the FNN model shown using voltage vs. State of charge graph rates by comparing the true and estimated values. Data used are for battery discharged cycled between 1.5–4.5 V vs. Na/Na^+ at a 0.05 C rate of (**a**) 2nd cycle, (**b**) 5th cycle, and (**c**) 10th cycle. Mean square value and R^2 value are mentioned on each graph.

4. Conclusions

In this paper, a two-layer feed forward neural network was designed with 15 neurons in each layer, and sigmoid was used as the activation function. The FNN mapped the measured battery signal voltage, current, and charge value directly to SOC and achieved the competitive estimation performance. The FNN can estimate the SOC for the highly non-linear nature of a Na-ion battery at different current rates (0.05 C, 0.1 C, 0.5 C, 1 C, 2 C), with R^2 value of ~0.97–0.99 and ~0.99 for higher cycle numbers, and a higher cutoff voltage of 4.5 V vs. Na^+/Na. The FNN can self-learn its weight using a gradient descent technique called Adam, and it was trained for 20 number of epochs. The future work is to train this model on the dataset of various drive cycles such as Urban Dynamometer Driving Schedule (UDDS), the Highway Fuel Economy Driving Schedule (HWFET), the Unified Driving Schedule (LA92), and the Supplemental Federal Test Procedures for various temperatures.

Author Contributions: D.D.: Formal analysis, Writing—original draft. I.B.: Formal analysis, Writing—reviewing and editing. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: The present work was carried out with material synthesis and physical characterization at SOLBAT-TTU Energy Research Laboratory, Tennessee Technological University, and electrochemical characterization at the Oak Ridge National Laboratory (ORNL), Oak Ridge, Tennessee. Darbar Devendrasinh acknowledges Jagjit Nanda (ORNL) for providing the laboratory facilities for performing the Electrochemical Analysis and Ethan C. Self (ORNL) for helping with the characterization.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. IEA—International Energy Agency. Global EV Outlook 2020. Available online: https://www.iea.org/reports/global-ev-outlook-2020 (accessed on 15 September 2020).
2. Plett, G.L. *Battery Management System Algorithms for HEV Battery Sate-of-Charge and State-of-Health Estimation. Advanced Materials and Methods for Lithium-ion Batteries*; Transworld Research Network: Kerala, India, 2007.
3. Plett, G.L. *Battery Management Systems, Volume I: Battery Modeling*; Artech House: Norwood, MA, USA, 2015.
4. Mu, H.; Xiong, R. Modeling, evaluation, and state estimation for batteries. In *Modeling, Dynamics, and Control of Electrified Vehicles*; Du, H., Cao, D., Zhang, H., Eds.; Woodhead Publishing: Duxford, UK, 2018; pp. 1–38.
5. Roscher, M.A.; Sauer, D.U. Dynamic electric behavior and open-circuit-voltage modeling of LiFePO 4-based lithium ion secondary batteries. *J. Power Sources* **2011**, *196*, 331–336. [CrossRef]
6. Ng, K.S.; Moo, C.-S.; Chen, Y.-P.; Hsieh, Y.-C. Enhanced coulomb counting method for estimating state-of-charge and state-of-health of lithium-ion batteries. *Appl. Energy* **2009**, *86*, 1506–1511. [CrossRef]
7. Waag, W.; Fleischer, C.; Sauer, D.U. Critical review of the methods for monitoring of lithium-ion batteries in electric and hybrid vehicles. *J. Power Sources* **2014**, *258*, 321–339. [CrossRef]
8. Hossain Lipu, M.S.; Hannan, M.A.; Hussain, A.; Ayob, A.; Saad, M.H.M.; Karim, T.F.; How, D.N.T. Data-driven state of charge estimation of lithium-ion batteries: Algorithms, implementation factors, limitations and future trends. *J. Clean. Prod.* **2020**, *277*, 124110. [CrossRef]
9. How, D.N.T.; Hannan, M.A.; Hossain Lipu, M.S.; Ker, P.J. State of charge estimation for lithium-ion batteries using model-based and data-driven methods: A review. *IEEE Access* **2019**, *7*, 136116–136136. [CrossRef]
10. Ng, M.-F.; Zhao, J.; Yan, Q.; Conduit, G.J.; Seh, Z.W. Predicting the state of charge and health of batteries using data-driven machine learning. *Nat. Mach. Intell.* **2020**, *2*, 161–170. [CrossRef]
11. Charkhgard, M.; Farrokhi, M. State-of-charge estimation for lithium-ion batteries using neural networks and EKF. *IEEE Trans. Ind. Electron.* **2010**, *57*, 4178–4187. [CrossRef]
12. Du, J.; Liu, Z.; Wang, Y. State of charge estimation for Li-ion battery based on model from extreme learning machine. *Control. Eng. Pract.* **2014**, *26*, 11–19. [CrossRef]
13. Tiwari, B.; Bhattacharya, I. State of charge estimation of sodium ion battery under different operating conditions using cascade forward backpropagation algorithm. In Proceedings of the 2018 IEEE Green Energy and Smart Systems Conference (IGESSC), Long Beach, CA, USA, 29–30 October 2018. [CrossRef]
14. Chemali, E.; Kollmeyer, P.J.; Preindl, M.; Emadi, A. State-of-charge estimation of Li-ion batteries using deep neural networks: A machine learning approach. *J. Power Sources* **2018**, *400*, 242–255. [CrossRef]
15. Banza Lubaba Nkulu, C.; Casas, L.; Haufroid, V.; De Putter, T.; Saenen, N.D.; Kayembe-Kitenge, T.; Musa Obadia, P.; Kyanika Wa Mukoma, D.; Lunda Ilunga, J.-M.; Nawrot, T.S.; et al. Sustainability of artisanal mining of cobalt in DR Congo. *Nat. Sustain.* **2018**, *1*, 495–504. [CrossRef]
16. Erdmann, L.; Graedel, T.E. Criticality of non-fuel minerals: A review of major approaches and analyses. *Environ. Sci. Technol.* **2011**, *45*, 7620–7630. [CrossRef] [PubMed]
17. Han, M.H.; Gonzalo, E.; Singh, G.; Rojo, T. A comprehensive review of sodium layered oxides: Powerful cathodes for Na-ion batteries. *Energy Environ. Sci.* **2015**, *8*, 81–102. [CrossRef]
18. Chakraverty, S.; Sahoo, D.M.; Mahato, N.R. McCulloch–Pitts neural network model. In *Concepts of Soft Computing: Fuzzy and ANN with Programming*; Springer: Singapore, 2019. [CrossRef]
19. Rosenblatt, F. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychol. Rev.* **1958**, *65*, 386–408. [CrossRef] [PubMed]
20. Amari, S.-I. Backpropagation and stochastic gradient descent method. *Neurocomputing* **1993**, *5*, 185–196. [CrossRef]
21. Learning PyTorch with Examples. Available online: https://pytorch.org/tutorials/beginner/pytorch_with_examples.html (accessed on 26 January 2021).
22. TensorFlow. Module:tf.compat.v1.train. Available online: https://www.tensorflow.org/api_docs/python/tf/compat/v1/train (accessed on 26 January 2021).
23. Hossain Lipu, M.S.; Hannan, M.A.; Hussain, A.; Ayob, A.; Saad, M.H.M.; Muttaqi, K.M. State of charge estimation in lithium-ion batteries: A neural network optimization approach. *Electronics* **2020**, *9*, 1546. [CrossRef]
24. Park, J.; Lee, J.; Kim, S.; Lee, I. Real-time state of charge estimation for each cell of lithium battery pack using neural networks. *Appl. Sci.* **2020**, *10*, 8644. [CrossRef]
25. sklearn.model_selection.GridSearchCV. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html (accessed on 26 January 2021).
26. sklearn.model_selection.RandomizedSearchCV. Available online: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html (accessed on 26 January 2021).
27. Kingma, D.P.; Lei Ba, J. Adam: A Method For Stochastic Optimization. *arXiv* **2017**, arXiv:1412.6980.
28. Krishna Kumar, S. On weight initialization in deep neural networks. *arXiv* **2017**, arXiv:1704.08863.

29. Caruana, R.; Lawrence, S.; Giles, L. Overfitting in neural nets: Backpropagation, conjugate gradient, and early stopping. In *Advances in Neural Information Processing Systems 13, Proceedings of the 13th International Conference on Neural Information Processing Systems, Denver, CO, USA, 27 November–2 December 2000*; MIT Press: Cambridge, MA, USA, 2001.
30. Cortes, C.; Mohri, M.; Rostamizadeh, A. L2 Regularization for Learning Kernels. *arXiv* **2012**, arXiv:1205.2653.
31. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
32. Darbar, D.; Muralidharan, N.; Hermann, R.P.; Nanda, J.; Bhattacharya, I. Evaluation of electrochemical performance and redox activity of Fe in Ti doped layered P_2-$Na_{0.67}Mn_{0.5}Fe_{0.5}O_2$ cathode for sodium ion batteries. *Electrochim. Acta* **2021**, *380*, 138156. [CrossRef]
33. Darbar, D.; Reddy, M.V.; Bhattacharya, I. Understanding the effect of Zn doping on stability of cobalt-free P_2-$Na_{0.60}Fe_{0.5}Mn_{0.5}O_2$ cathode for sodium ion batteries. *Electrochem* **2021**, *2*, 323–334. [CrossRef]